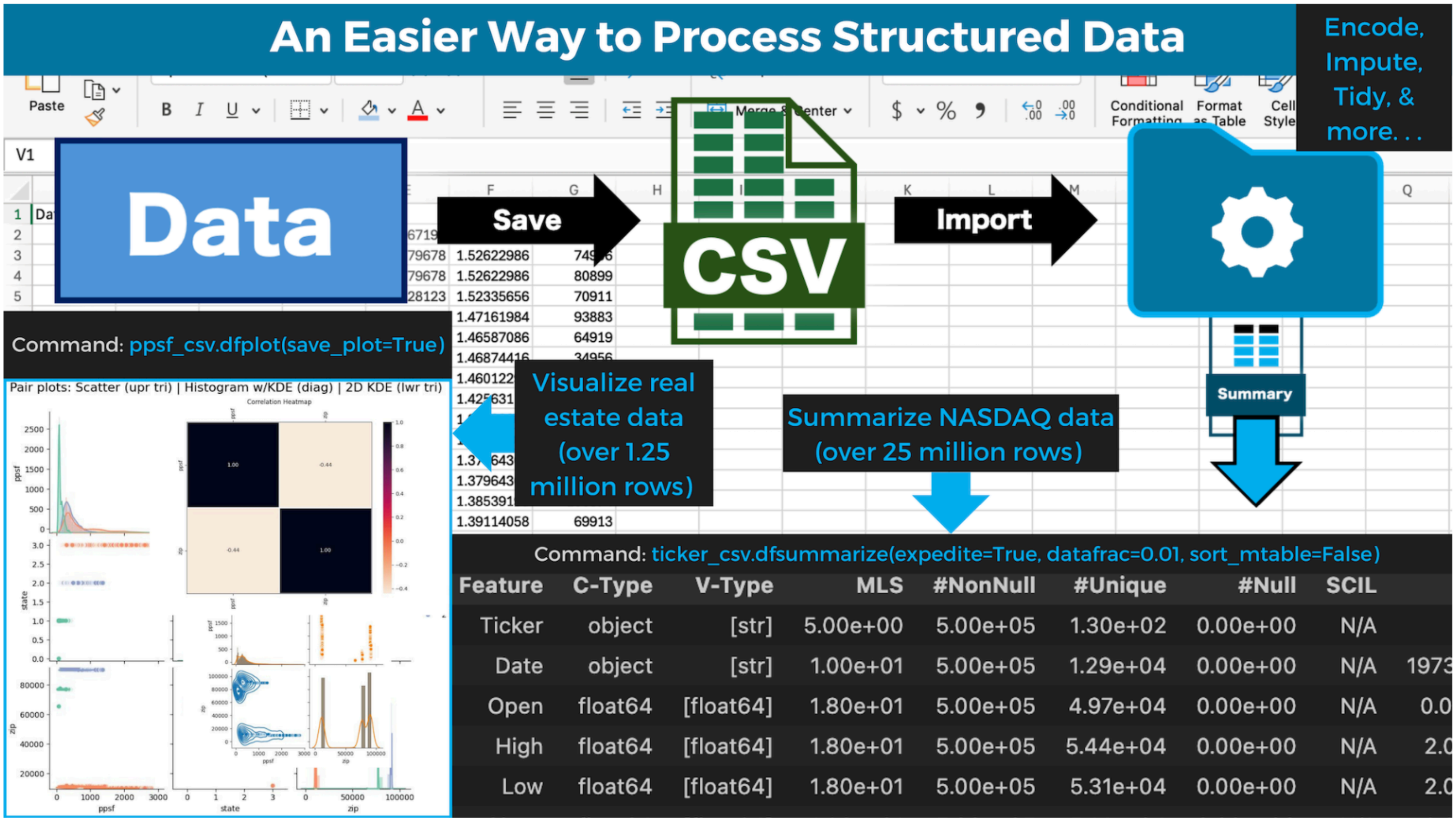


Built on top of the Pandas library, this utility extends the functionality and enhances the efficiency of Pandas to streamline workflows. By enabling workflow automation and optimization, it aims to simplify and accelerate data processing tasks. These optimized processes reduce manual effort, minimize human error, enhance consistency, and boost productivity for data scientists and analysts.

The constituent modules are organized into packages that provide specific functionalities.

Package	Content and purpose
🗁 encoding	Code to perform encoding of categorical and ordinal data in a dataframe.
🗁 graphing	Code to visualize data in a dataframe.
🗁 imputation	Code to impute missing values in a dataframe.
🗁 summary	Code to summarize a dataframe.
🗁 tidying	Code to transform and clean raw data in a dataframe.
🗁 time	Code to generate and treat temporal features in a dataframe.
🗁 utilities	Code to help implement primary functions.

The modules (associated with each package) and their corresponding brief descriptions are provided below.

Package	Module	Description
encoding	dfencode.py	Performs encoding of categorical data (as in sklearn label encoding and one-hot encoding) in the source dataframe. This module, unlike sklearn label encoding, is not susceptible to imputing non-existing relationships stemming from the inherent rank ordering of categories.
graphing	dfbcat_algo.py	Allows for the determination of categorical columns from features that would ordinarily be delineated as numeric data type.
graphing	dfplot_corr.py	Generates a correlation heatmap for both numerical and categorical data types.
graphing	dfplot_line.py	Generates a customizable lineplot from the given data.
graphing	dfplot_mcat.py	Generates multiple categorical plots.
graphing	dfplot_pair.py	Generates a pairplot from the source dataframe.
graphing	dfplot_scatter.py	Generates a customizable scatterplot from the given data.
graphing	dfplot.py	Simplifies the accurate visualization of data in a source dataframe through feature data type identification and conversion that is critical for delineating feature associations in the given data.
imputation	dfavg.py	Returns an individual estimate or a series of approximations for the imputation of missing values in the designated column of the source dataframe.
imputation	dfknn.py	Returns numerical and non-numerical missing value estimates based on k-nearest neighbors in a given dataframe column.
imputation	dfmid.py	Returns the mid value (median) for numeric and string data columns in the source dataframe.
imputation	dfmle.py	Returns numerical and non-numerical missing value estimates using machine learning (random forest).
imputation	dfsce.py	Returns numerical and non-numerical missing value estimates using user-guided input based on the data source.
summary	dfquery.py	Handles queries and search operators (i.e., '==', '>', '>=', '<', '<=', '!=', '.isin') for querying and filtering dataframes.
summary	dfsummarize.py	Summarizes information per column in the source dataframe. Summary includes a listing of column names, data types, unique value counts, null and non-null value counts, row indexes per column where null values appear, and the stats array among other metadata.
summary	dftopcount.py	Computes individual frequencies (counts) for the top 'n' values (where, n is user-defined) as well as the corresponding percentage contributions (shares) of the selected values to the total value count per column of the source dataframe.

summary	dfunique.py	dataframe.
tidying	dfdetouts.py	Detects (using a number of approaches, including isolation forest and ensembling) and removes outliers per column in the source dataframe.
tidying	dfindexes.py	Extracts the row indexes (index) that correspond(s) to a user-defined value in the given column of the source dataframe.
tidying	dforder.py	Re-arranges (sorts) columns, row values (by column), or row indexes as well as yields a subset of the source dataframe along either axis.
tidying	dfrename.py	Renames column and index labels of the source dataframe.
tidying	dfreplace.py	Replaces values of a target column in the source dataframe based on user-designated value(s) or value(s) of one or more of the other columns in the dataframe.
tidying	dfscale.py	Scales numerical data by column or across the entire source dataframe in one shot using one of three scaling options.
tidying	dfsearch.py	Searches the input data source for a string from a given list.
tidying	dftidy.py	Tidies the given data, including removing duplicate rows and user-defined columns, searching and replacing aberrant and missing values, encoding categorical and ordinal variables, generating temporal features, scaling data, and performing sanity checks.
tidying	dftodtype.py	Converts the given source dataframe column to the desired data type if possible.
time	dfcycdate.py	Creates cyclical temporal components from source data comprising datetime series.
time	dfsplitdate.py	Splits a datetime column in the source dataframe into constituent temporal datetime components.
time	dftime.py	Creates and handles temporal features in the source dataframe.
utilities	calc_3m.py	Computes the 3m's (min, median, and max) of the stats array extracted from the non-object data type columns of the source dataframe.
utilities	calc_avg_days.py	Calculates an average cyclical period (in days per month) for a given Pandas series in the source dataframe.
utilities	calc_cyclicals.py	Calculates cyclical temporal features (i.e., sin and cos representations) of the given time series in the source dataframe.
utilities	cat_corr.py	Assesses the association between two categorical variables (based on Cramer's V statistic which includes a correction originating in the Bergsma-2013 paper).
utilities	check_dtypes.py	Checks whether the user input comprising one or more values conforms to one or more data types.
utilities	check_input.py	Checks whether the user input and its constituent elements conform to the corresponding data types.
utilities	check_ls_in_ls.py	Checks whether the user input and its constituent elements conform to the data type 'list' and the given list element data types, respectively, and whether one list is subsumed by the other.
utilities	check_range.py	Checks and resolves raw data that violate the designated min-max, low-high, or other data range comprising polarized-value pairs.
utilities	combine.py	Generates an element index collection from an input iterable with or without fixing one of the iterable elements.
utilities	generate_ltr_ls.py	Creates a string list that may include the entire alphabet and alphabet combinations comprising lower- and upper-case letters.

utilities	get_missing_indexes.py	Extracts the indexes of source list elements not in the target list, with the option of inserting a user- designated element in the target list.
utilities	idx_to_col.py	Converts the given index collection to the corresponding column label grouping(s).
utilities	indexes.py	Obtains the indexes (index) that correspond(s) to a user-defined value in a list.
utilities	ls_concat.py	Concatenates or flattens a list of lists.
utilities	ls_depth.py	Gets list depth by counting the number of opening square brackets.
utilities	ls_insert.py	Inserts a user-specified element into the target list at the positions indicated by the index list.
utilities	map_lists.py	Maps two lists of equal lengths to a list of their corresponding element pairs and returns the elements of one list from the corresponding matched elements of the other.
utilities	parse_range.py	Parses the given min-max, low-high, or other range comprising polarized-value pairs.
utilities	pattern_in_str.py	Validates (or invalidates) the given pattern in the string input.
utilities	permute.py	Generates permutations of elements in an input iterable according to the length of permuted lists fixed by the parameter 'r' if 'r' is specified or the length of the iterable itself if 'r' is missing.
utilities	pluralize.py	Generates the correct word variant for an underlying countable noun.
utilities	return_single_or_multi_input.py	Returns the input as a valid single entity or as valid multiple entities in the form of a list.
utilities	sci_to_std.py	Converts a number in scientific notation to standard notation.
utilities	set_axis_range.py	Sets limits to values and formats tick labels along the x and the y axes.
utilities	show_err_msg.py	Logs error messages to the console.
utilities	std_to_sci.py	Converts a number in standard notation to scientific notation.
utilities	time_func.py	Clocks wall and CPU times for the designated function.
utilities	time_str_to_24hr.py	Converts a time string into a 24-hour format (military time), wherein the date component can be expressed in a user-specified valid format.
utilities	time_str_to_date.py	Converts a time string into a date.
utilities	to_df.py	Concatenates lists along the given axis to return a dataframe comprising the original lists.
utilities	to_dict.py	Adds or replaces key-value pairs in a dictionary.
utilities	val_date.py	Validates and converts date strings.
utilities	zscore_iter.py	Iterates over a list of z-scores corresponding to a list of values in a context-aware manner and identifies values associated with z-scores that do not fall in the open interval (-zthresh, +zthresh), where 'zthresh' is the user-defined z-score threshold.

> ## Leave the heavy lifting to our Pandas utility

Summarize

Summarizing data in a dataframe can be more informative and relevant than Pandas describe() and info() functions for a majority of use cases. Abstracting metadata from a dataframe with our Pandas utility enables a high-level data view that includes the following

- dataframe feature (column) name(s).
- data type per column.
- date type of column values.
- maximum length of column values as strings.
- number of unique values (null values are excluded).
- number of non-null values.
- number of null (missing) values.
- row indexes per column where null values appear.
- dataframe column search for user-defined value and row index(es) where value is found.
- counts and percentage shares of total count of top 'n' values by column.
- summary stats (min, max, median, potential outlier values, and variance inflation factors) for numerical features.
- values with user-defined rounding precision.

Problem/ use case:

Summarize the small *California's Vehicle Fuel Type Count by Zip Code* data set from Google Cloud Storage using our Pandas utility, showing the stats array (default=True) and the null indexes (rows with missing values) by column in the main table (mtab). Find and replace any null value variants of the type in ['NaN', 'nan', 'unknown', 'NA', 'na', 'N/A', '', ' '] with np.nan at the outset. Do not specify datetime and numerical columns, but let the algorithm attempt to identify columns that potentially possess the corresponding individual patterns. Additionally, display the frequency table (ftab) comprising top-3 counts and percentage shares for each column. Sort the frequency table, ordering 'Col' and 'Count' in the ascending and descending order, respectively.

Code:

```
import pandas as pd
from <pandas-utility>.summary import dfsummarize

dftp         = pd.read_csv('../data/transport/untidy_vehicle_data_toy.csv')
dftp['Date'] = pd.to_datetime(dftp['Date'])
mtab, ftab   = dftp.dfsummarize(show_null_index_col=True, freq_table=True, sort_ftable=True)
```

```
*** Finding and replacing potential missing value variants in the dataframe...
*** The dataframe has no missing value variants of the type: ['NaN', 'nan', 'unknown', 'NA', 'na', 'N/A', 'M', '', ' '].
*** Computing frequencies and proportions in the frequency table for the input dataframe...
    Dataframe shape (r, c): (499, 7)
    Features (cols) list  : ['Date', 'Zip Code', 'Model Year', 'Fuel', 'Make', 'Light_Duty', 'Vehicles']
```



```
*** Searching data for missing values...
```

```
*** Getting ready to check for datetime pattern in ['Date', 'Zip Code', 'Model Year', 'Fuel', 'Make', 'Light_Duty', 'Vehicles'].
>>> Checking for datetime pattern in 'Date' using %Y-%m-%d as the date format...
    Done in 0.02 sec.
```

```
*** Parsing and converting numeric columns...
*** Numerical data type columns that comprise data with a likely numerical pattern: ['Zip Code', 'Model Year', 'Fuel', 'Make', 'Light_Duty', 'Vehicles'].
*** Getting ready to compute the stats array...
*** Using ['Zip Code', 'Vehicles'] as the list of numerical columns for the stats array.
*** Eliminating any residual missing values for computing the stats array...
*** Parsing numerical data for potential outlier values (POVs)...
*** Done POVs!
*** Attempting to compute variance inflation factors (VIFs)...

Legend   :
   Ctype : Column data type
   Vtype : Column value data type
   MLS   : Maximum length of column value as string
   Mid   : Median
   POV   : Potential outlier value(s) for zthresh=1.5
   VIF   : Variance inflation factor(s)
```

	Feature	Ctype	Vtype	MLS	#Unique	#Non-null	#Null	NullIndex	Min	Mid	Max	POV	VIF
0	Date	datetime64[ns]	[datetime64]	29	130	497	2	[2, 41]	2018/10/01	2018/12/16	2019/03/08	NaN	NaN
1	Zip Code	float64	[float64]	7	3	497	2	[1, 19]	90000.0	90001.0	90002.0	[90000.0, 90002.0]	1.094
2	Model Year	object	[float, str]	5	15	497	2	[2, 41]	NaN	NaN	NaN	NaN	NaN
3	Fuel	object	[float, str]	24	8	497	2	[19, 58]	NaN	NaN	NaN	NaN	NaN
4	Make	object	[float, str]	9	43	496	3	[1, 19, 58]	NaN	NaN	NaN	NaN	NaN
5	Light_Duty	object	[float, str]	3	2	496	3	[0, 41, 58]	NaN	NaN	NaN	NaN	NaN
6	Vehicles	float64	[float64]	6	151	496	3	[2, 19, 41]	1.0	25.0	3178.0	[1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0, ...	1.094

	Col	Value	Count	Share
0	Date	2018-10-01 00:00:00	8	0.0161
1	Date	2018-10-26 00:00:00	6	0.0121
2	Date	2018-10-20 00:00:00	6	0.0121
3	Fuel	Gasoline	336	0.6800
4	Fuel	Diesel and Diesel Hybrid	55	0.1100
5	Fuel	Flex-Fuel	54	0.1100
6	Light_Duty	Yes	435	0.8800
7	Light_Duty	No	61	0.1200
8	Make	OTHER/UNK	129	0.2600
9	Make	Type_A	37	0.0700
10	Make	Type_J	35	0.0700
11	Model Year	<2006	81	0.1600
12	Model Year	2007	53	0.1100
13	Model Year	2008	45	0.0900
14	Vehicles	13.0	24	0.0484
15	Vehicles	14.0	24	0.0484
16	Vehicles	16.0	20	0.0403
17	Zip Code	90001.0	361	0.7300
18	Zip Code	90002.0	120	0.2400
19	Zip Code	90000.0	15	0.0300

Problem/ use case:

Summarize the large *Consumer Complaint Data* set (> 4.7M rows) using our Pandas utility, showing the stats array and the null indexes by column in the main table (mtab) without dropping rows containing missing data (dropna=False). Provide datetime and numerical column names or targets ('Date received', 'Date sent to company', 'Complaint ID') to the algorithm. To save time, avoid checking the target columns for bool data type (check_bool=False). Display a frequency table (ftab) comprising top-3 counts and percentage shares for each column. Sort the frequency table, ordering 'Col' and 'Count' in the ascending and descending order, respectively.

Code:

```
import pandas as pd
from <pandas-utility>.summary import dfsummarize

dfcc = pd.read_csv('../datasets/complaints.csv', dtype=np.object_)
mtab, ftab   = dfcc.dfsummarize(
                                dropna=False,
                                show_null_index_col=True,
                                stats=True,
                                freq_table=True,
                                sort_ftable=True,
                                check_bool=False,
```

```
)
```

```
*** Finding and replacing potential missing value variants in the dataframe...
*** The dataframe has no missing value variants of the type: ['NaN', 'nan', 'unknown', 'NA', 'na', 'N/A', '', ' '].
*** Computing frequencies and proportions in the frequency table for the input dataframe...
    Dataframe shape (r, c): (4700472, 18)
    Features (cols) list  : ['Date received', 'Product', 'Sub-product', 'Issue', 'Sub-issue', 'Consumer complaint narrative', 'Company public response', 'Company', 'State', 'ZIP code',
    Missing value count   : 15645844
    All null row indexes  : [ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19]...

*** Getting ready to check for datetime pattern in ['Date received', 'Date sent to company'].
>>> Checking for datetime pattern in 'Date received' using %Y-%m-%d as the date format...
    Done in 155.27 sec.
>>> Checking for datetime pattern in 'Date sent to company' using %Y-%m-%d as the date format...
    Done in 161.39 sec.
```

:   :

	Feature	Ctype	Vtype	MLS	#Unique	#Non-null	#Null	NullIndex	Min	Mid	Max	POV	VIF
0	Date received	object	[str]	10	4458	4700472	0	[]	2011/12/01	2021/11/16	2024/02/15	NaN	NaN
1	Product	object	[str]	76	21	4700472	0	[]	NaN	NaN	NaN	NaN	NaN
2	Sub-product	object	[float, str]	48	87	4465182	235290	[101760, 109720, 118301, 129588, 131360, 13164...	NaN	NaN	NaN	NaN	NaN
3	Issue	object	[str]	80	177	4700472	0	[]	NaN	NaN	NaN	NaN	NaN
4	Sub-issue	object	[float, str]	145	273	3972708	727764	[45, 118, 132, 168, 192, 200, 287, 358, 406, 5...	NaN	NaN	NaN	NaN	NaN
5	Consumer complaint narrative	object	[float, str]	32763	1393636	1687705	3012767	[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13,...	NaN	NaN	NaN	NaN	NaN
6	Company public response	object	[float, str]	119	12	2225489	2474983	[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13,...	NaN	NaN	NaN	NaN	NaN
7	Company	object	[str]	88	7111	4700472	0	[]	NaN	NaN	NaN	NaN	NaN
8	State	object	[float, str]	36	64	4656164	44308	[664, 936, 1618, 3413, 3502, 5640, 6358, 8147,...	NaN	NaN	NaN	NaN	NaN
9	ZIP code	object	[float, str]	5	33405	4670249	30223	[153077, 160122, 160207, 162389, 167089, 16815...	NaN	NaN	NaN	NaN	NaN
10	Tags	object	[float, str]	29	4	461039	4239433	[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13,...	NaN	NaN	NaN	NaN	NaN
11	Consumer consent provided?	object	[float, str]	20	5	3751560	948912	[0, 1, 2, 3, 5, 6, 7, 8, 11, 12, 13, 15, 16, 1...	NaN	NaN	NaN	NaN	NaN
12	Submitted via	object	[str]	12	7	4700472	0	[]	NaN	NaN	NaN	NaN	NaN
13	Date sent to company	object	[str]	10	4407	4700472	0	[]	2011/12/01	2021/11/17	2024/02/15	NaN	NaN
14	Company response to consumer	object	[float, str]	31	9	4700464	8	[577924, 1333695, 1588562, 1590660, 1758285, 1...	NaN	NaN	NaN	NaN	NaN
15	Timely response?	object	[str]	3	2	4700472	0	[]	NaN	NaN	NaN	NaN	NaN
16	Consumer disputed?	object	[float, str]	3	3	768316	3932156	[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13,...	NaN	NaN	NaN	NaN	NaN
17	Complaint ID	object	[str]	7	4700472	4700472	0	[]	1.0	4917570.5	8356602.0	[1, 5, 7, 16, 20, 22, 24, 26, 27, 36, 37, 39, ...	NaN

	Col	Value	Count	Share
0	Company	EQUIFAX, INC.	944326.0	0.2000
1	Company	TRANSUNION INTERMEDIATE HOLDINGS, INC.	863797.0	0.1800
2	Company	Experian Information Solutions Inc.	789646.0	0.1700
3	Company public response	Company has responded to the consumer and the ...	1984771.0	0.8900
4	Company public response	Company believes it acted appropriately as aut...	137384.0	0.0600
5	Company public response	Company chooses not to provide a public response	52473.0	0.0200
6	Company response to consumer	Closed with explanation	3252416.0	0.6900
7	Company response to consumer	Closed with non-monetary relief	1079152.0	0.2300
8	Company response to consumer	In progress	169625.0	0.0400
9	Complaint ID	8273071	1.0	0.0000
10	Complaint ID	5533888	1.0	0.0000
11	Complaint ID	5100953	1.0	0.0000
12	Consumer complaint narrative	TLDR	NaN	NaN
13	Consumer consent provided?	Consent not provided	1837162.0	0.4900
14	Consumer consent provided?	Consent provided	1689023.0	0.4500
15	Consumer consent provided?	Other	217024.0	0.0600
16	Consumer disputed?	No	619938.0	0.8100
17	Consumer disputed?	Yes	148378.0	0.1900
18	Date received	2024-01-18	7113.0	0.0015
19	Date received	2024-01-23	6840.0	0.0015
20	Date received	2024-01-24	6576.0	0.0014
21	Date sent to company	2024-01-18	7161.0	0.0015
22	Date sent to company	2024-01-23	6888.0	0.0015
23	Date sent to company	2024-01-24	6629.0	0.0014
24	Issue	Incorrect information on your report	1318094.0	0.2800
25	Issue	Improper use of your report	679905.0	0.1400
26	Issue	Problem with a credit reporting company's inve...	589338.0	0.1300
27	Product	Credit reporting, credit repair services, or o...	2163883.0	0.4600
28	Product	Credit reporting or other personal consumer re...	570472.0	0.1200
29	Product	Debt collection	535760.0	0.1100
30	State	FL	557040.0	0.1200
31	State	CA	543324.0	0.1200
32	State	TX	492703.0	0.1100
33	Sub-issue	Information belongs to someone else	873784.0	0.2200
34	Sub-issue	Reporting company used your report improperly	447633.0	0.1100
35	Sub-issue	Their investigation did not fix an error on yo...	410555.0	0.1000
36	Sub-product	Credit reporting	2710082.0	0.6100
37	Sub-product	Checking account	219423.0	0.0500
38	Sub-product	General-purpose credit card or charge card	183635.0	0.0400
39	Submitted via	Web	4159632.0	0.8800
40	Submitted via	Referral	245764.0	0.0500
41	Submitted via	Phone	175200.0	0.0400
42	Tags	Servicemember	269708.0	0.5900
43	Tags	Older American	153666.0	0.3300

46	Timely response?	No	[illegible]	[illegible]
47	ZIP code	XXXXX	112147.0	0.0240
48	ZIP code	30349	8515.0	0.0018
49	ZIP code	19143	6723.0	0.0014

Transform & Clean

Transforming and cleaning ("tidying") source dataframe includes the following operations.

- returning original dataframe if no transformation is required.
- removing user-defined columns from source dataframe.
- removing duplicate rows.
- renaming features (columns).
- searching and replacing missing value characters (entire dataframe).
- replacing aberrant value or removing row(s) containing the aberration(s) (by column).
- excluding user-defined columns from the null replacement process.
- replacing or removing null values.
- replacing values in a target column contingent on the value(s) in the source column(s).
- encoding categorical and ordinal variables.
- converting column data (e.g., boolean values) into numeric data type.
- converting date feature(s).
- generating temporal features.
- selecting and sorting columns or passing user-defined column order.
- performing range check on boolean, datetime, and numerical columns.
- detecting and removing outliers.
- scaling data.
- rounding numbers with user-defined rounding precision.
- saving tidied data.

Problem/ use case: Given the dataframe 'dfz' ...

```
dfz         = pd.DataFrame([5, 10, 15, np.nan, 'NaN', 20, 10, 10, 0, 'M', 5],
                             columns=['Alpha'])
dfz['Date'] = [np.nan, np.nan, '2023-01-01', '2023-02-01', '2023-03-01', '2023-04-01',
               '2023-05-01', '2023-06-01', '2023-07-01', '2023-08-01', np.nan]
```

... perform the following operations.

- remove duplicate rows.
- search for missing value variants and replace them with np.nan.
- encode categorical candidates (columns).
- convert 'Date' column from object to datetime data type.
- split 'Date' column into temporal elements, including weekly and quarterly time components.
- generate temporal features based on the 'Date' column.
- identify and remove outliers.
- scale numerical columns using the sklearn robust scaler.
- round dataframe numerical values to 3 decimal places.
- save the 'tidied' dataframe in a csv file.

Code:

```
from <pandas-utility>.tidying import dftidy
dfz.dftidy(auto=True)
```

```
seed        : 100
*** Done removing 1 duplicate row corresponding to the original row index in [10].
*** Row count has dropped from 11 to 10 as a result of removing row copies.
*** Done replacing missing value variants in ['NaN', 'nan', 'unknown', 'NA', 'na', 'N/A', 'M', '', ' '] with np.nan. Total replacements made: 2.
search_col  : None
search_str  : None
replace_val : None
*** Done converting column 'Alpha' to numeric dtype.
An exception of type ValueError occurred (Unable to parse string "2023-01-01" at position 2)...Skipping conversion of column 'Date' to numeric dtype...
null counts : [3, 2]
```

```
1   Date   mode  2023-01-01
*** Proceeding to treat null values...
*** Done replacing null values in dataframe.
*** Categorical column determination of features based on pcat_ceil suggests columns in ['Alpha'] as likely categorical features.
*** Datetime pattern found in 'Date'. This column will be excluded by dfbcat_algo.
*** Categorical column determination of features based on bcat_algo recalibration suggests columns in [] as likely categorical features.
*** That none of the features is categorical is more likely than not.
*** Checking for datetime pattern in column data before encoding...
*** Column that comprises data with a likely datetime pattern: ['Date'].
*** Done converting column 'Date' to datetime dtype.
enc_type: cat
*** Warning! Perform encoding only after null values in the dataframe have been treated.
*** Warning! Since no columns are specified, encoding is proceeding autonomously.
*** Encoding process completed.
*** Done splitting date col(s) and generating cyclicals.
   Alpha     Date_m_sin     Date_m_cos  Date_W_sin  Date_W_cos
0    5.0  0.000000e+00  1.000000e+00    0.000000    1.000000
1   10.0  0.000000e+00  1.000000e+00    0.000000    1.000000
2   15.0  0.000000e+00  1.000000e+00    0.000000    1.000000
3   10.0  5.000000e-01  8.660254e-01    0.464723    0.885456
4   10.0  8.660254e-01  5.000000e-01    0.822984    0.568065
5   20.0  1.000000e+00  6.123234e-17    0.992709    0.120537
6   10.0  8.660254e-01 -5.000000e-01    0.885456   -0.464723
7   10.0  5.000000e-01 -8.660254e-01    0.568065   -0.822984
8    0.0  1.224647e-16 -1.000000e+00    0.120537   -0.992709
9   10.0 -5.000000e-01 -8.660254e-01   -0.464723   -0.885456
*** Proceeding with outlier detection using ensemble method...
out_ls : ['Alpha', 'Date_m_sin', 'Date_m_cos', 'Date_W_sin', 'Date_W_cos']
idx_ls : [[0, 8], [9], [], [4, 5, 6], [8]]
*** Found no common row indexes across outliers.
Scaler: RobustScaler (rs)
*** Done scaling the numerical col(s) in the dataframe.
*** Done saving to csv file: 2024-02-25_07-45-45_tidy_data.csv.
```

	Alpha	Date_m_sin	Date_m_cos	Date_W_sin	Date_W_cos
0	-5.0	-0.323	0.431	-0.385	0.385
1	0.0	-0.323	0.431	-0.385	0.385
2	5.0	-0.323	0.431	-0.385	0.385
3	0.0	0.323	0.354	0.227	0.317
4	0.0	0.795	0.144	0.699	0.131
5	10.0	0.968	-0.144	0.922	-0.131
6	0.0	0.795	-0.431	0.781	-0.475
7	0.0	0.323	-0.641	0.363	-0.685
8	-10.0	-0.323	-0.718	-0.227	-0.784
9	0.0	-0.968	-0.641	-0.997	-0.721

Visualize

Visualizing ("graphing") data with our Pandas utility is intuitive and requires no prior knowledge of Python graphing tools. It is as simple as passing a Pandas dataframe, generated when loading a seaborn example dataset or reading a comma-separated values (csv) file using Pandas, as an argument to dfplot().

Problem/ use case: Given the dataframe 'dfz' ...

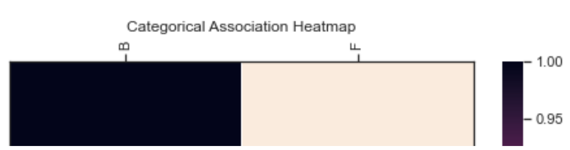
```
dfz      = pd.DataFrame([5, 10, 15, 20, 10, 10, 0, 5], columns=['A'])
dfz['B'] = [1, 2, 3, 1, 3, 4, 5, 2]
dfz['C'] = [1, 2, 3, 4, 5, 1, 1, 2]
dfz['D'] = ['2023-01-01', '2023-02-01', '2023-03-01', '2023-04-01',
            '2023-05-01', '2023-06-01', '2023-07-01', '2023-08-01']
dfz['E'] = ['0', '1', '2', '3', '4', '5', '6', '7']
dfz['F'] = ['a', 'a', 'b', 'a', 'b', 'b', 'c', 'b']
```

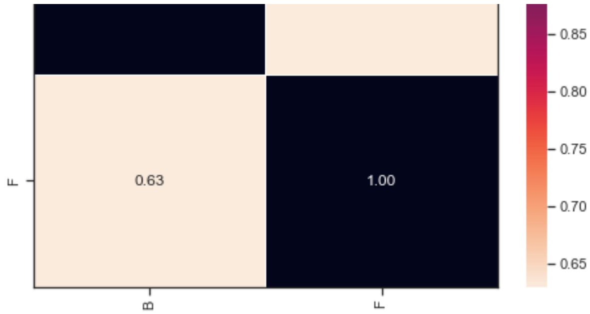
... visualize the data in dfz.

Code:

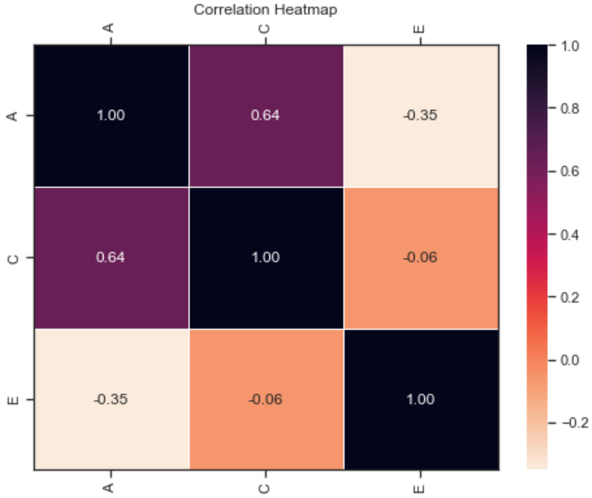
```
from <pandas-utility>.graphing import dfplot
dfplot(dfz)
```

Examples of graphical output from dfplot()

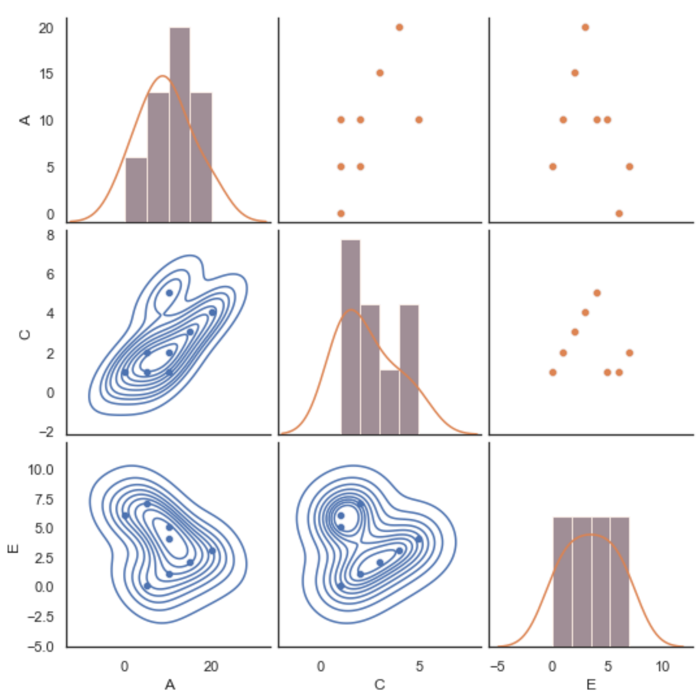




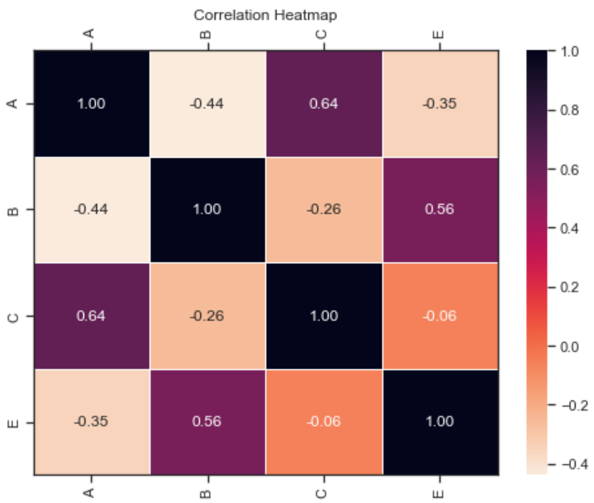
Correlation Heatmap



Pair plots: Scatter (upr tri) | Histogram w/KDE (diag) | 2D KDE (lwr tri)



```
dfplot(dfz, bcat_algo=False)
```



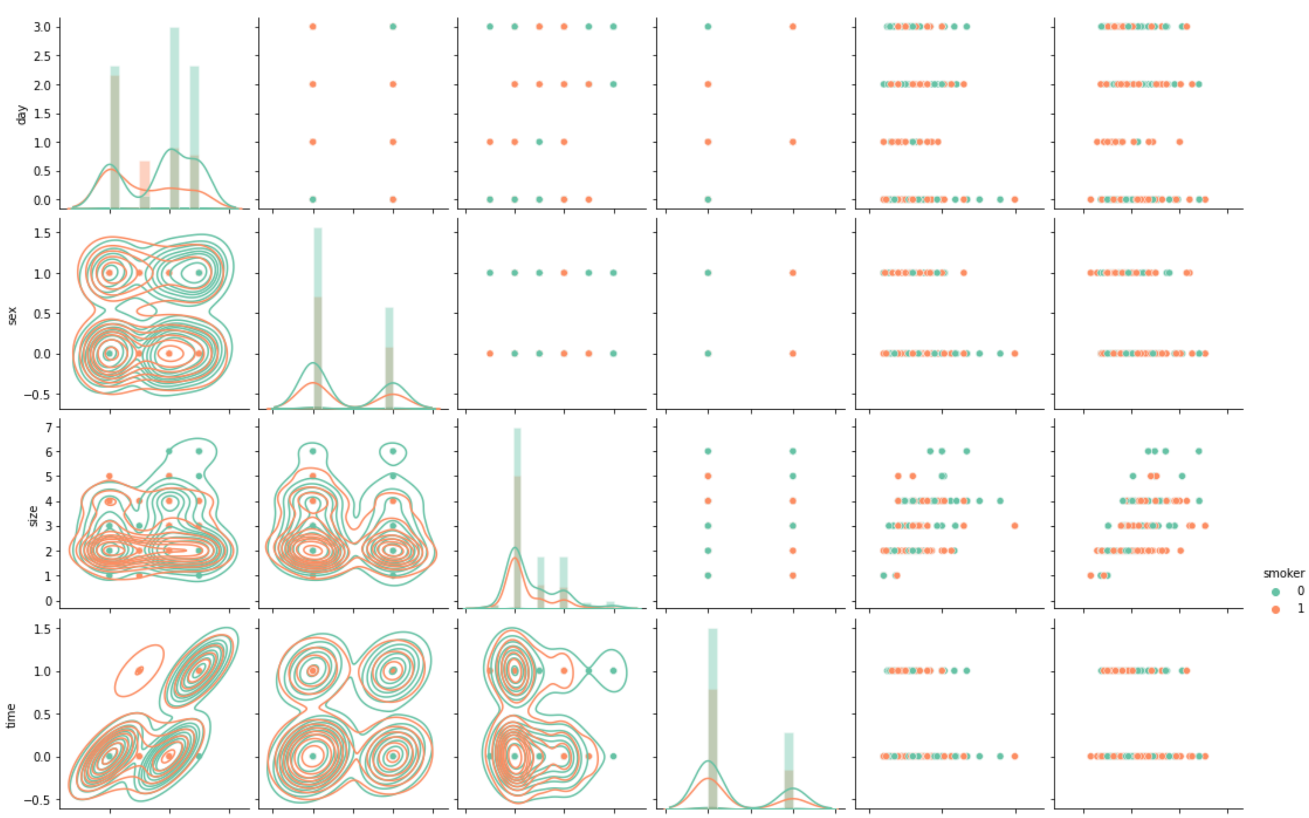
Problem/ use case: Visualize the Seaborn 'tips' dataset.

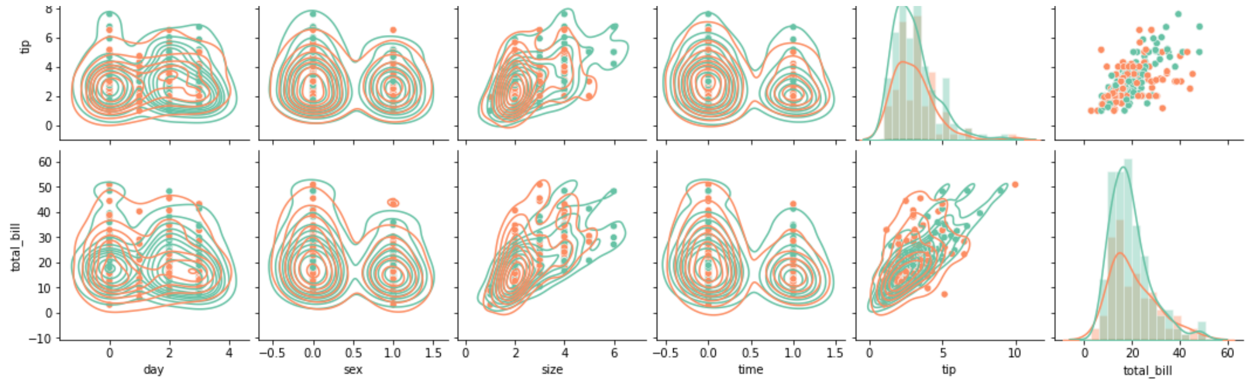
Code:

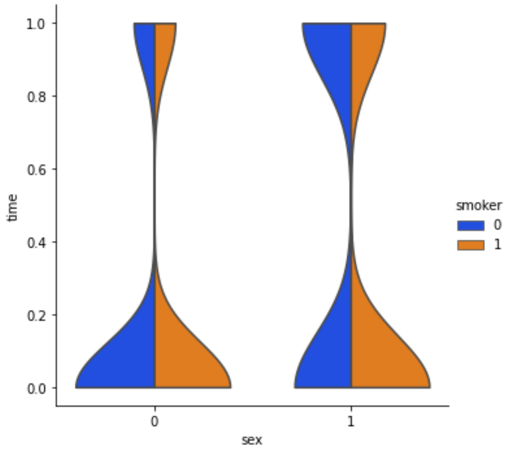
```
import seaborn as sns
from <pandas-utility>.graphing import dfplot
tips = sns.load_dataset('tips')
dfplot(tips)
```

Examples of graphical output from dfplot() and dfplot_mcat()

Pair plots: Scatter (upr tri) | Histogram w/KDE (diag) | 2D KDE (lwr tri)



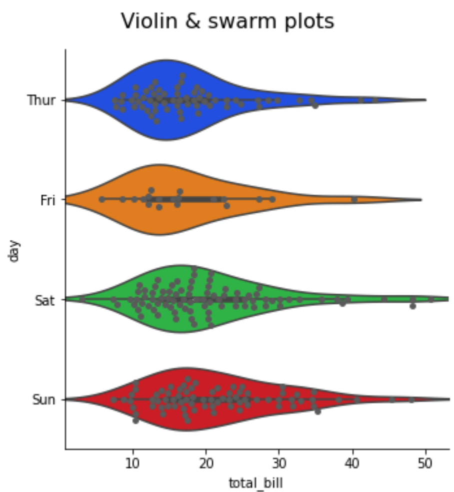




Problem/ use case: Visualize the distribution of total bill across days in the Seaborn 'tips' dataset.

Code:

```
from <pandas-utility>.graphing import dfplot_mcat
dfplot_mcat(tips, 'total_bill', 'day', aux_plot_kind='swarm')
```



Other attributions:

- Folder icon source: [Freepik]
- Tips dataset: [Seaborn]